BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to the field of graphical user interfaces (GUIs), and more particularly to a system and method for managing a scalable list of items for display in a display device of a small footprint device.

2. Description of the Related Art

Graphical user interfaces (GUIs) enable users to interact with software applications in an intuitive manner, utilizing graphical components or controls, such as buttons, input text fields, menus, etc. Although graphical user interfaces are often associated with software applications that run on general computing devices, such as desktop computers, GUIs are becoming more common in other types of computing devices, such as small footprint devices.

The field of "smart" small footprint devices is growing and changing rapidly. Small footprint devices include handheld computers, personal data assistants (PDAs), cellular phones, global positioning system (GPS) receivers, game consoles, set top boxes, and many more such devices. Small footprint devices are becoming increasingly powerful and enabled to run software applications or services historically associated with general computing devices, such as desktop computers. For example, many small footprint devices are now able to run web browser applications.

As small footprint devices have become more powerful and the software applications running on small footprint devices have become more complex, it has become increasingly feasible and desirable to enable users to easily interact with the applications through graphical user interfaces. However, experience has shown that many techniques commonly used when implementing a GUI intended to run in a resource-rich environment such as a desktop computer have various drawbacks if applied to a GUI intended to run in a small footprint device. Small footprint devices may have very strong resource constraints, such as the amount of memory and the processing power available. Thus, it is often necessary or desirable to design specialized or optimized user interface components that perform well in the environment of a small footprint device.

One type of user interface component or control often supported by graphical user interface systems is a list component or control for displaying a list of items. In various systems or applications, items in a list component may represent any of various types of entities and may be displayed in various ways, e.g., through the use of text strings, icons, background shading, etc., or through various combinations of these. For example, an email client program may use a list component to display a list of e-mail messages, where each item or row in the list represents a particular e-mail message and may be displayed with various icons to indicate the status of the message, such as whether it has already been read, etc.

Such GUI list components are familiar to users of desktop computers. As described above, it may also be desirable to use such a rich GUI list component within the environment of a small footprint device. However, previous approaches to implementing GUI list components often suffer from various drawbacks when the list components are used within a small footprint device environment.

One potential drawback of many previous approaches to implementing GUI list components is that the list components may allocate memory for maintaining information associated with each list item. For example, as each data item is inserted into a list, a list component may create and maintain various user interface structures or other data structures associated with the data item. The use of the information maintained by the list component for each list item may have various benefits, such as making it possible to display the list items faster as a user traverses through the list. However, in a memory-constrained environment, the disadvantage of the extra memory required for each list item may outweigh any potential benefits. Thus, it may be desirable to provide a system and method for displaying list items in a scalable manner, such that, as many items are added to the list, the memory requirements of the list component itself remain constant.

Another potential drawback of some previous approaches to implementing GUI list components is that the list components often provide no control or only limited control over the display or appearance of each list item. For example, a programmer may be limited to setting certain application programming interface (API) attributes controlling the appearance of each list item, while it may be desirable to customize various other aspects of list item display that are unsupported by the API. Also, a list component may be limited in the types of data items or objects it is able to display. It may thus be desirable to provide a system and method for displaying list items such that the rendering or display of each list item may be easily controlled by a client programmer, e.g., to control the way in which a programmer-defined object is displayed to the user.

SUMMARY OF THE INVENTION

The problems outlined above may in large part be solved by a system and method for managing a scalable list of items for display in a display device of a small footprint device, as described herein. A client program running in a small footprint device may instantiate a "list container object". The list container object may provide an API enabling the client program to then add "list item data objects" to the list container object. The list container object may maintain these list item data objects in any of various ways as appropriate to a particular implementation or programming environment, e.g., as an item element vector, array, linked list, etc.

The list container object may instantiate a fixed number of "item renderer objects", which are responsible for appropriately displaying the list item data objects. Each item renderer object may correspond to a row in the displayed list. The list container object interfaces with the set of item renderer objects, in order to manage the display of the list. In one embodiment, the item renderer objects are instances of a class that implements an "item renderer interface". The item renderer interface may include methods, such as SetData(), GetData(), etc. for interacting with the item renderer objects. Thus, a general framework is described in which any of various types of objects may be displayed, by implementing the item renderer interface methods appropriately for different item renderer object implementations.

The list container object may keep track of the set of list item data objects being displayed at any given time. In one embodiment, the list container object simply maintains a "start index" specifying which item is currently displayed in the first row of the list. As a user interacts with the list, e.g., by scrolling up or down, the list container object may receive user interface events indicating the user's action, may determine the new start index for items to display, and may instruct each item renderer object to redisplay the appropriate list item data object, e.g., by calling the SetData() item renderer interface method for each item renderer object, passing the corresponding list item data object as a parameter.

BRIEF DESCRIPTION OF THE DRAWINGS

Other objects and advantages of the invention will become apparent upon reading the following detailed description and upon reference to the accompanying drawings in which:

Figure 1 is a block diagram illustrating one embodiment of a system supporting a scalable user interface list of items;

Figure 2 is a block diagram illustrating one embodiment of a framework for providing a scalable user interface list of items;

Figure 3 illustrates one embodiment of an item renderer interface implemented by list item renderer objects;

Figure 4 illustrates the management of portions of the user interface display by user interface components associated with list item renderer objects;

Figure 5 is a screen shot illustrating an example of user interface list implemented as described herein;

Figure 6 is a flowchart diagram illustrating one embodiment of a method of creating a list for display in a user interface in accordance with the framework herein; and

Figure 7 is a flowchart diagram illustrating one embodiment of managing the dynamic operation of a user interface list.

While the invention is susceptible to various modifications and alternative forms, specific embodiments thereof are shown by way of example in the drawings and will be described herein in detail. It should be understood, however, that the drawings and detailed description thereto are not intended to limit the invention to the particular form disclosed, but on the contrary, the intention is to cover all modifications, equivalents and alternatives falling within the spirit and scope of the present invention as defined by the appended claims.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Figure 1 – Exemplary System

Figure 1 is a block diagram illustrating one embodiment of a system 720 supporting a scalable user interface list of items. The system 720 may be employed in any of various types of computing devices, such as desktop computers or workstations, small footprint devices, etc.

As shown in Figure 1, the system 720 may include a processor 710. The processor 710 may be any of various types, including an x86 processor, e.g., a Pentium class, a PowerPC processor, a CPU from the SPARC family of RISC processors, as well as others. In various embodiments, such as an embodiment in which the system 720 illustrates a small footprint device, the processor 710 may be a less powerful processor than those listed above or may be a processor developed specifically for a small footprint device, such as a digital signal processor (DSP). The processor 710 may have various clock speeds, including clock speeds similar to those found in desktop computer-class processors, as well as lower speeds such as 16 MHz.

The system 720 also includes a memory 712 coupled to the processor 710. The memory 712 may comprise any of various types of memory, including DRAM, SRAM, EDO RAM, etc., or a non-volatile memory such as a magnetic media, e.g., a hard drive, or optical storage. The memory 712 may comprise other types of memory as well, or combinations thereof. For an embodiment in which the system 720 illustrates a small footprint device, the memory 712 may have a very small storage capacity compared to a typical desktop computer system.

As shown in Figure 1, the memory 712 may store code 724 for implementing a scalable list of user interface items. The user interface list code 724 may comprise code, e.g. class definitions or modules, for implementing various objects used in managing the user interface list, such as described below.

The memory 712 may also store a client program 718 that uses the user interface list code 724. For example, the client program 718 may include program instructions for instantiating objects of the user interface list code 724 and invoking methods on the objects, as described below. It is noted that the user interface list code 724 may be tightly coupled with the client program 718. For example, the client program 718 may be an application program, and the user interface list code 724 may be a portion of the code base of the application. However, in other embodiments, the user interface list code 724 may be loosely coupled with the client program 718. For example, the user interface list code may be included in a user interface library for use by any of various client programs.

As shown in Figure 1, the system 720 may also comprise a display 716. The display 716 may be any of various types, such as an LCD (liquid crystal display), a CRT (cathode ray tube) display, etc. It is noted that the display for a typical small footprint device, such as a smart cellular phone, may be small compared to the display of a desktop computer system. The display 716 is provided to display various information, including lists generated in accordance with the below description.

As shown in Figure 1, the system 720 may also comprise an input mechanism 714. The input mechanism 714 may be any of various types, as appropriate for a particular system. For example, the input mechanism may be a keyboard, mouse, trackball, touch pen, microphone, modem, infrared receiver, etc.

As noted above, in various embodiments, the system 720 may be illustrative of a small footprint device. As used herein, a small footprint device is a hardware device comprising computing resources such as a processor and a system memory, but having significantly greater constraints on one or more of these resources than a typical desktop computer has. For example, a typical small footprint device may have two megabytes of memory or less, whereas a typical desktop system may have 64 megabytes or more. Also a typical small footprint device may have significantly less processing power than a typical desktop computing system, either in terms of processor type, or processor speed, or both. For example, a particular personal data assistant device may have a 16 MHz processor, whereas a typical desktop system may have a processor speed of 100 MHz or higher. Also, a typical small footprint device may have a display size significantly smaller than the display screen of a desktop computing system. For example, the display screen of a handheld computer is typically small compared to the display screen of a desktop monitor. It is noted that the specific numbers given above are exemplary only and are used for comparison purposes.

Small footprint devices may also have constraints on other resource types compared to typical desktop computing systems, besides the memory, processor, and display size resources described above. For example, a typical small footprint device may not have a hard disk, may not have a network connection, or may have an intermittent network connection, or may have a wireless network connection, etc.

Many small footprint devices are portable and/or are small compared to desktop computers, but are not necessarily so. Also, many small footprint devices are primarily or exclusively battery-operated. Also, small footprint devices may typically have a more limited or narrow range of usage possibilities than a typical desktop computing system. Thus, in various embodiments, the system illustrated in Figure 1 is illustrative of, but is not limited to, any one of the following: handheld computers, wearable devices (e.g., wristwatch computers), personal data assistants (PDAs), "smart" cellular telephones, set-top boxes, game consoles, global positioning system (GPS) units, electronic textbook devices, etc. Since new classes of consumer devices are rapidly emerging, it is not possible to provide an exhaustive list of small footprint devices. However, the term "small footprint device" is intended to include such devices as may reasonably be included within the spirit and scope of the term as described above.

It is noted that in various embodiments the system and method described herein may also be employed in a general-purpose computer system, such as a desktop PC or mainframe computer.

Figures 2 – 3: Framework for Providing a Scalable User Interface List of Items

Figure 2 is a block diagram illustrating one embodiment of a framework for providing a scalable user interface list of items. As described above, the illustrated framework may be implemented within a small footprint device.

The framework illustrated in Figure 2 includes a list container object 100. A client program may instantiate a list container object 100 and add list item data objects 101 to the list container object 100 via an application programming interface (API) provided by the list container object. The client program may be a client program written using any of various programming languages or development environments. The list container object 100 may be an object implemented in any of various ways and may provide the API to client programs in any of various ways, e.g., as object methods that client programs may invoke. For example, in an embodiment implemented using the Java™ programming language, the list container object may provide a method with a signature such as:

```
public synchronized void addItem (Object item)
```

Client programs may then call the addItem() method to add list item data objects 101 to the list container 100. The list container may store or maintain the list item data objects 101 in any of various ways, e.g., as a vector, an array, a linked list, etc.

As shown in Figure 2, at any given time, only a subset of the list item data objects 101 may actually be displayed. The number of items that can be displayed at a given time depends on the size of the list, i.e., the number of "rows" the list container object is configured with. Configuring the list container object with a number of rows is discussed below. For example, if the list container object maintains ten list item data objects but is only configured to have four rows (as illustrated in Figure 2), then only a subset of the list item data objects may be displayed. The list container object may maintain a "start index" that specifies which of the list item data objects 101 is currently displayed in the first row of the list.

As shown in Figure 2, the list container object 100 interfaces with a set of list item renderer objects 102A-102D (referred to collectively as list item renderer objects 102). Each list item renderer object 102 may correspond to a row in the displayed list. For example, in Figure 2, list item renderer object 102C may correspond to the third row. Each list item renderer object 102 implements code for appropriately displaying a list item data object. The list container object 100 interacts with the list item renderer objects 102 to manage the display of the list.

In one embodiment, the list item renderer objects may be objects including a "set data" method with a signature such as:

```
public void SetData (Object data)
```

Thus, the list container object may simply use the start index to retrieve the appropriate list item data object corresponding to each list item renderer object and pass the list item data object to the SetData() method for the list item renderer object. For example, in Figure 2, the list container object may pass the fifth list item data object shown in the figure to the list item renderer object 102A, may pass the sixth list item data object shown in the figure to the list item renderer object 102B, etc.

In response to the SetData() method being invoked, the list item renderer objects may display the received list item data object in the user interface, as appropriate for a particular type of list item data object. For example, for list item data objects representing e-mail messages, the list item renderer objects may retrieve information from the message objects, such as the sender's name, the message subject, etc., and may then display this information in the user interface.

In one embodiment, an "item renderer" interface is defined, and list item renderer objects designed to display list item data objects of various types are all expected to implement the item renderer interface. One embodiment of an item renderer interface implemented by list item renderer objects constructed using the Java™ programming language is shown in Figure 3.

As shown in Figure 3, the item renderer interface may include the SetData() method discussed above, as well as additional methods, such as getData(), setVisible(), isVisible(), getRenderComponent(), etc.

By standardizing the interface between the list container object and the item renderer objects, the Figure 2 framework may advantageously be used to display any of various types of list item data objects. The list container may manage the list item data objects and the list item renderer objects independently of the type of list item data objects being displayed, and the type-specific logic may be encapsulated within particular list item renderer object implementations. It is noted that the encapsulation of the type-specific logic within list item renderer objects may offer a program developer an easy way to have full control over how list item data objects are displayed.

The appropriate class for the list item renderer objects may be specified or determined in various ways. In one embodiment, it is the responsibility of the client program to inform the list container object of the appropriate list item renderer class, given the types of the list item data objects. For example, if the client program inserts objects representing e-mail messages into the list container, then the client program may inform the list container object that the appropriate class for the list item renderer objects is a class named, for example, MessageItemRenderer, where the MessageItemRenderer class implements the item renderer interface and comprises code for retrieving information from e-mail message objects and displaying the information. The client program may inform the list container object of the appropriate list item renderer class in any of various ways. For example, the list container object may include a SetRendererClass() method callable by client programs. Embodiments are contemplated in which the list item renderer class may be determined in various other ways, e.g., by determining the class based on information in the list item data objects themselves.

Figure 4 – User Interface Component Usage

List item renderer objects may implement the list item user interface display logic in any of various ways. In one embodiment, the list item renderer objects may utilize pre-existing user interface components or controls in displaying information, e.g., by instantiating user interface objects or inheriting from user interface classes. For example, in an embodiment implemented using the Java™ programming language, list item renderer objects may inherit from a standard Java™ Abstract Window Toolkit (AWT) user interface component, such as a java.awt.Canvas component. Thus, what appears in the user interface display as an integrated user interface list component may actually comprise multiple user interface components.

List item renderer objects may of course utilize any of various types of user interface components as necessary, depending on the type of information to be displayed, the particular user interface components supported for a particular system, etc. In response to the list container object setting the data for the list item renderer objects, each list item renderer object may interface with its associated user interface component to display the data appropriately, e.g., by invoking various paint or draw methods on the user interface component, etc.

As described above, each list item renderer object may correspond to a particular row in the displayed list. Thus, the size of the user interface component associated with each list item renderer object may be set to a size appropriate for controlling an area of the user interface display that corresponds to a particular row of the displayed list. For example, Figure 4 illustrates a portion of a user interface display that is divided into four rectangular areas, each corresponding to a list row. Each of the areas may be managed by a user interface component associated with a list item renderer object. For example, in Figure 4, user interface component 152A may be associated with the list item renderer object 102A of Figure 2, user interface component 152B may be associated with the list item renderer object 102B of Figure 2, etc.

The list item renderer objects may enable a client program to obtain the associated user interface components. For example, the item renderer interface embodiment illustrated in Figure 3 shows a getRendererComponent() method for obtaining the user interface component associated with a list item renderer object.

As shown in Figure 4, a list container user interface component 150 may be associated with the list container object, the size of which is set to the size of the entire user interface area covered by the displayed list.

The size of the list and the list rows and the number of rows in the list may be determined in any of various ways. For example, when instantiating a list container object, a client program may specify this information, e.g., by passing the list size and the number of rows as parameters to a constructor function.

Note that the use of pre-existing user interface components or controls may enable the list container to respond to various types of user interface events supported by a particular environment or user interface component. For example, user interface events, such as keypresses, mouse clicks, etc., may be brokered from the list container user interface component 150 to the list container object, enabling the list container object to cause the displayed list to scroll, etc.

Although the above description discusses the display of list item data via the use of pre-existing user interface components or controls, the list item renderer objects may of course implement all of the user interface display logic themselves, if desirable.

Figure 5 – Exemplary List Display

As discussed above, the system and method described herein may be applied to display any of various types of data items in a user interface list. Figure 5 is a screen shot illustrating one such user interface list. Figure 5 illustrates a list with eight rows, each of which displays header information for an email message, such as the name of the sender, the date sent, the size of the message, etc.

The Figure 5 list appears similar to many common user interface list components. However, the Figure 5 list is implemented and operates as discussed herein.

Figure 6 – Creating a List for Display in a User Interface

Figure 6 is a flowchart diagram illustrating one embodiment of a method of creating a list for display in a user interface in accordance with the framework described above with reference to Figures 2 and 3.

In step 200, a client program instantiates a list container object. Step 200 may be accomplished in any of various ways depending on a particular implementation and programming environment.

In step 202, the client program adds list item data objects to the list container object, as discussed above.

In step 204, the client program specifies an appropriate class for the list item renderer objects to instantiate. As discussed above, the list item renderer objects should be of a class enabled to display the particular list item data objects added in step 202. The client program may specify the list item renderer class, e.g., by calling a SetRendererClass() method of the list container object.

In step 206, the list container object instantiates the appropriate number of list item renderer objects of the class specified in step 204. As discussed above, the number of list item renderer objects may correspond to the number of list rows displayed in the user interface. In step 206, the list container object may also set the corresponding data object for each list item renderer object. As discussed above, the list container may simply call a SetData() method of each list item renderer object, passing the appropriate list item data object as a parameter. For initial display, the list item data objects passed to the list item renderer objects would typically correspond to the list item data objects at the beginning of the list. For example, for a list with four visible rows, the first four list item data objects may be passed to the list item renderer objects.

In step 208, each list item renderer object displays its corresponding list item data object received in step 206. As discussed above, the list item data objects may be displayed in any of various ways, as appropriate to a particular type of list item data object, application, development platform, etc.

In step 210, the list container object begins processing events, such as user interface events. For example, the list container object may receive and process events representing various user actions, such as scrolling through the list. For such an event, the list container may then update the list display as described below.

As noted above, Figure 6 represents one embodiment of a method of creating a list for display in a user interface, and various steps of Figure 6 may be added, omitted, combined, altered, performed in different orders, etc. For example, Figure 6 is discussed in terms of adding list item data objects to the list container object in a single step. However, the list item data objects may of course be added to the list container at any of various points during operation of the list.

As another example, the class determination of the list item renderer objects and the list item renderer object instantiation steps may be performed in various ways other than shown in Figure 6. For example, the list container object may interface with another object that determines the appropriate class of list item renderer objects to instantiate, manages the object instantiations, etc.

Figure 7 – Dynamic Operation of List

Figure 7 is a flowchart diagram illustrating one embodiment of managing the dynamic operation of a user interface list.

In step 300, the user changes the current list position. The user may change the current list position in any of various ways as supported by the user interface and the list container object. For example, as shown in the Figure 5 example, the user interface may include buttons such as the "Up" and "Down" buttons shown. When a user clicks one of these buttons, the client program may be operable to instruct the list container to move up or down. For example, in one embodiment the list container may provide methods such as:

```
public synchronized void scrollPageUp()
public synchronized void scrollPageDown()
```

The client program may then call these list container methods as appropriate.

The list container object itself may also be configured to receive and respond to various user interface events. For example, as discussed above, the list container object may be based on a user interface object that supports these capabilities.

In step 302, the list container object determines the new start index for the list, based on the action performed in step 300. As discussed above with reference to Figure 2, the list container may maintain a start index that specifies the first list item data object being displayed at any given time. In step 302, the list container object may update this start index as appropriate. For example, if the user scrolled down a page in step 300, then the list container may simply increase the start index, e.g., by adding the number of list rows displayed to the start index.

Once the start index has been updated, the list rows are re-displayed in step 304. The list container object may simply use the start index to retrieve the corresponding list item data object for each list item renderer object, and then inform the list item renderer object of the new data to be displayed, e.g., by calling the renderer object's SetData() method as discussed above.

In step 306, each list item renderer object displays its updated list item data object, similarly as discussed above.

As noted above, Figure 7 represents one embodiment of a method for managing the dynamic operation of the list, and various steps of Figure 7 may be added, omitted, combined, altered, performed in different orders, etc. For example, step 300 of Figure 7 is discussed in terms of a user initiating a change in the current list position. Such a change in the list position may of course also be caused programmatically.

Although the embodiments above have been described in considerable detail, numerous variations and modifications will become apparent to those skilled in the art once the above disclosure is fully appreciated. It is intended that the following claims be interpreted to embrace all such variations and modifications.